ＢＣＲＬ７８１０４　マイコン開発セット　マニュアル

第１版２０１４．１．１３　　　　**第１版**

【　製品概要　】

本マニュアルはＢＣＲＬ７８１０４　ＣＰＵボードのソフトウエア開発を行うために必要なソフトウエアインストゥール手順、添付ＣＤのサンプルプログラムの動作について解説されています。特に新しい統合開発環境ＣｕｂｅＳｕｉｔｅ+（ＣＳ+）における開発方法について多く記述してあります。
※本ＣＰＵボード開発にはルネサスエレクトロニクス社製Ｅ１が必要です。



１．開発環境、事前準備

１－１．　開発環境

ａ：開発セット　同梱物
ｂ：ＢＣＲＬ７８１０４　ＣＰＵボードの特徴
ｃ：Ｅ１エミュレータ（デバッカ）
ｄ：無償のＣｕｂｅＳｕｉｔｅ＋、ＲＬ７８用Ｃコンパイラのダウンロード
ｅ：ＣＤコピー、デバイスドライバ
ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較
　ｆ－１：ポートアクセス速度の比較
　ｆ－２：乗除演算速度の比較

１－２　動作、デバック

ａ：ＣｕｂｅＳｕｉｔｅ＋起動、コンパイル、書き込み、動作
ｂ：新しいプログラムを作る　ＣｕｂｅＳｕｉｔｅ+　操作
　ｂ－１：Ａ／Ｄ設計上の注意点
　ｂ－２：自動生成されたプログラム
　ｂ－３：Ｅ１から電源供給
　ｂ－４：コード生成後の初期値の変更
　ｂ－５：変数を見る
　ｂ－６：変数変化を実行中に確認する

２．サンプルプログラム

２－１．　ｓａｍｐｌｅ１　出力ポートのＯＮ，ＯＦＦ
２－２．　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ）、ＥＥＰＲＯＭ読み書き
２－３．　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力
２－４．　ｓａｍｐｌｅ４　割り込み
２－５．　ｓａｍｐｌｅ５　ＰＷＭ出力
２－６．　ｓａｍｐｌｅ６　三角、対数、平方根関数を使う
２－７．　ｓａｍｐｌｅ７　Ｄ／Ａコンバータ　sin、cos 値を出力してみる

１－１．開発環境

ａ：開発セット同梱物
ＢＣＲＬ７８１０４　ＣＰＵボード
ＣＤ（サンプルプログラム、ドキュメント）
マニュアル（本誌）
電源ケーブル、Ｖケーブル、ＵＳＢミニケーブル



※開発に必要なルネサスエレクトロニクス社製エミュレータ１は同封されておりません。別途必要です。

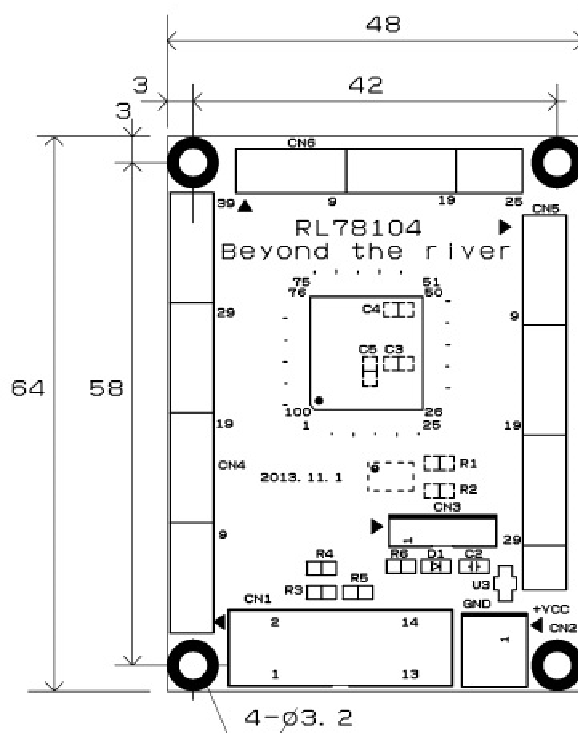
ｂ：ＢＣＲＬ７８１０４　ＣＰＵボードの特徴

●高性能、低消費電力、低コストな新設計ＲＬ７８コアを使用。４４ＤＭＩＰＳ/３２ＭＨｚ、６６μＡ／ＭＨｚ。３２ＭＨｚ±１%の高精度内蔵オシレータ　※１
●ＲＬ７８/Ｇ１４（Ｒ５Ｆ１０４ＰＪ）は幅広い動作電圧、周波数、低消費電力を実現した新世代汎用マイクロコンピュータです。様々な周辺機能（２０ｃｈ　Ａ／Ｄコンバータ、２ｃｈ　Ｄ／Ａコンバータ、４ｃｈ　ＵＡＲＴ、高性能ＰWMタイマ、ＬＩＮ－ｂｕｓ、Ｉ²Ｃ通信機能等）搭載。１００ピン。
●内蔵高速オシレータ　　３２ＭＨｚ（２．７～５．５Ｖ）。最小命令実行時間３１．２５ｎｓｅｃ。
●内蔵低速オシレーター　１５ＫＨｚ（ＴＹＰ）　ＣＰＵクロックとしては使用不可。
●メモリ容量　フラッシュＲＯＭ２５６Ｋバイト、ＲＡＭ２４Ｋバイト、データフラッシュ８Ｋバイト。電源を切ってもデータが保持されるＥＥＰＲＯＭ　２５ＬＣ２５６（容量３２、７６８ＢＹＴＥ　２００年以上データ保持）搭載　ライブラリ添付。
●基板大きさ、小型６４×４８×１５mm
●動作電圧電流　３．３Ｖ～５．５Ｖ、７．５mＡ　ＴＹＰＥ(５Ｖ、３２ＭＨｚ動作時)
最低１．６Ｖから動作可能（低電圧メインモード）
●豊富な周辺機能
Ｉ／Ｏポート　合計９２本（オープンドレイン、プルアップ指定可能）
Ａ／Ｄコンバータ：１０ビット分解能　２０ｃｈ
Ｄ／Ａコンバータ：８ビット分解能　２ｃｈ
ＵＡＲＴ：４ｃｈ
Ｉ²Ｃ：８ｃｈ（１ｃｈはＬＩＮ－ｂｕｓ通信対応）
タイマ：１６ビット　１２h、ウオッチドグタイマ、１２ビットリアルタイムクロック、インターバルタイマ内蔵
●乗除算・積和演算命令に対応、オンチップデバック機能内蔵
●シリアルコネクタでＶケーブルを接続し、ＵＳＢ使用可能。ミニＢコネクタ、ドライバＩＣ　ＦＴＤＩ社　ＶケーブルはＦＴ２３２ＲＬ搭載。

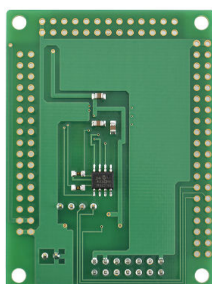
●エミュレータＥ１によるデバック用コネクタ搭載。Ｃ言語による１行実行、ブレークポイント、変数参照等可能です。

※１　速度比較は本マニュアル　１－１　ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較をご参照下さい。

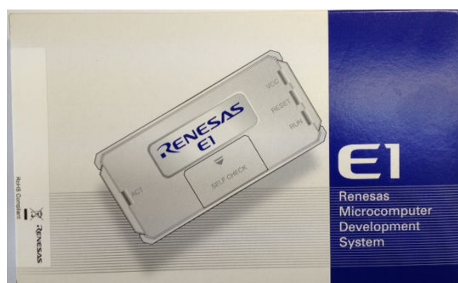
基板大きさ（部品面）



２５ＬＣ２５６は裏面搭載。



ｃ：Ｅ１エミュレータ



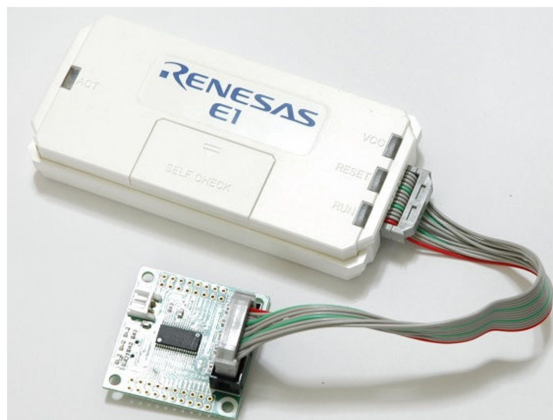
概要

Ｅ１エミュレータは、ルネサス主要マイコンに対応したオンチップデバッギングエミュレータです。基本的なデバッグ機能を有した低価格の購入しやすい開発ツールで、フラッシュプログラマとしても使用可能です。
Ｃ言語ソースデバックが可能で、１行実行、ブレークポイント設定、変数、レジスタ、メモリ参照等々、従来であれば高価なＩＣＥ（インサーキットエミュレータ）しか出来なかった機能が、安価に実現されています。また、使い方もＨＥＷ（統合開発環境）のＥ８ａと同じで、経験があれば半日で、無くても１日で必要な操作を会得することが出来ると思います。

マイコンとの通信として、シリアル接続方式とＪＴＡＧ接続方式の２種類に対応しています。使用可能なデバッグインタフェースは、ご使用になるマイコンにより異なります。
また、基本デバッグ機能に加え、ホットプラグイン機能（動作中のユーザシステムに後から E1 エミュレータを接続して、プログラムの動作確認を行うことが可能）を搭載しているため、プログラムのデバッグ・性能評価に大きく貢献できます。

対応ＭＰＵ

- V850 ファミリ
- RX ファミリ
- RL78 ファミリ
- R8C ファミリ
- 78K ファミリ



Ｅ１を購入するとＣＤが添付されていて、ドライバーのインストールとセルフチェックを行った後に、ネットから開発環境ＣｕｂｅＳｕｉｔｅ＋とＣコンパイラのダウンロードを行います。

ｄ：無償版ＲＬ７８用Ｃコンパイラのダウンロード
　プログラムの開発はルネサスエレクトロニクス社の統合開発環境ＣｕｂｅＳｕｉｔｅ+でＣ言語を用い動作させることができます。ＣＤ添付のサンプルプログラムはこの環境下で作成されています。無償版をダウンロードして使用します。
　ネット検索で→「ＲＬ７８　ＣｕｂｅＳｕｉｔｅ＋　コンパイラダウンロード」の検索で表示されます。

統合開発環境	【無償評価版】統合開発環境 CubeSuite+ V2.00.00a (分割ダウンロード版)	Apr.23.13	CubeSuite+パッケージです。デバッガおよび無償評価版コンパイラも含まれます。アップデートにも使用できます。 対応マイコン: V850、RXファミリ、RL78ファミリ、78K0、78K0R	
統合開発環境	【無償評価版】統合開発環境 CubeSuite+ V2.00.00a (一括ダウンロード版)	Apr.23.13	CubeSuite+パッケージです。デバッガおよび無償評価版コンパイラも含まれます。アップデートにも使用できます。 対応マイコン: V850、RXファミリ、RL78ファミリ、78K0、78K0R	

統合開発環境とＣコンパイラが同時にダウンロードされます。

以下省略

ｆ：ＲＬ７８とＨ８／３００Ｈ、Ｒ８Ｃの速度比較

ＲＬ７８は、有名なＨ８／３０４８の代わりに検討される方も多いと思われますが、実行速度はどうなのでしょうか？　開発環境を含めて以前より進化していなければ使う意味がないとお考えの方も多いかと思われます。

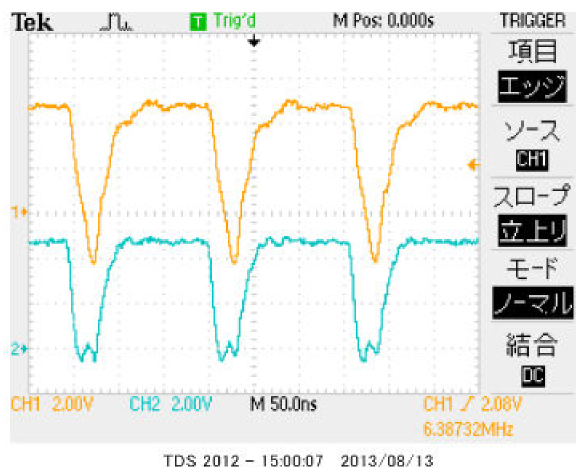
ｆ－１　ポートアクセス速度比較

単純なポートアクセスプログラムで比較してみます。
ＲＬ７８のポートを１，０繰り返すプログラムです。

```
                void main(void)
                {
00187               R_MAIN_UserInit();
                    /* Start user code. Do not edit comment generated here */
                    while (1U)
                    {
0018b                   P2 = 0x00;
0018d                   P2 = 0xff;

00190               }
                    /* End user code. Do not edit comment generated here */
00192           }
```

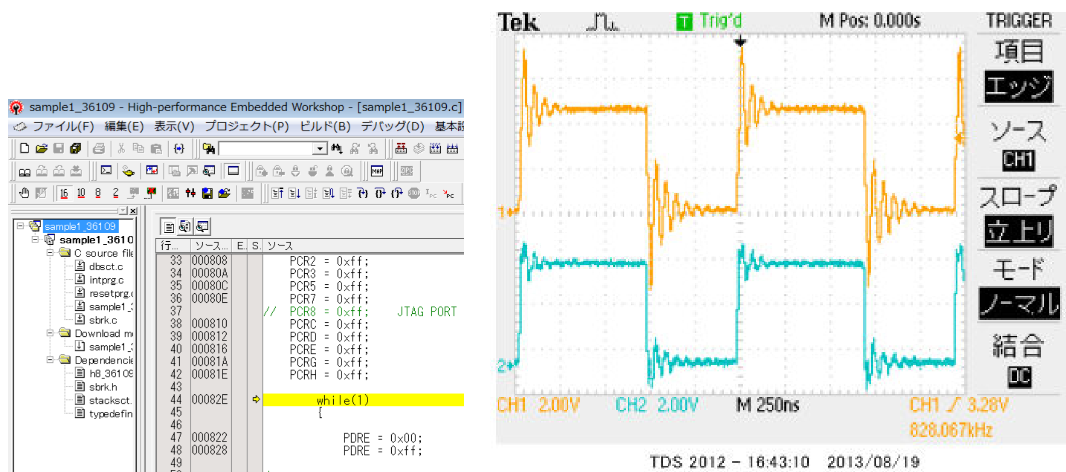
オシロスコープでＰ２０、Ｐ２１波形を観測すると６．３８７３２ＭＨｚという周波数でポートの１，０を繰り返すことが分かります。（クロック３２ＭＨｚ）



TDS 2012 - 15:00:07　2013/08/13

この命令の詳細は
ｗｈｉｌｅ（１Ｕ）
｛
Ｐ２　＝　０ｘ００；　　　//ポートを０にする
Ｐ２　＝　０ｘｆｆ；　　　//ポートを１にする
｝　　　　　　　　　　　　//上行にジャンプする
という３つの動作を行っています。波形が１から０に落ちて、上がる手前の時間が１命令の実行時間です。波形上約３０ｎｓｅｃ程度なので、カタログ値　３１．２５ｎｓｅｃと大きく相違は無いように思います。１クロックで１命令実行はＲＩＳＣ並みですね。１の時間が０に比べて長いのはポートを１にする、上行にジャンプするの２命令実行しているからです。

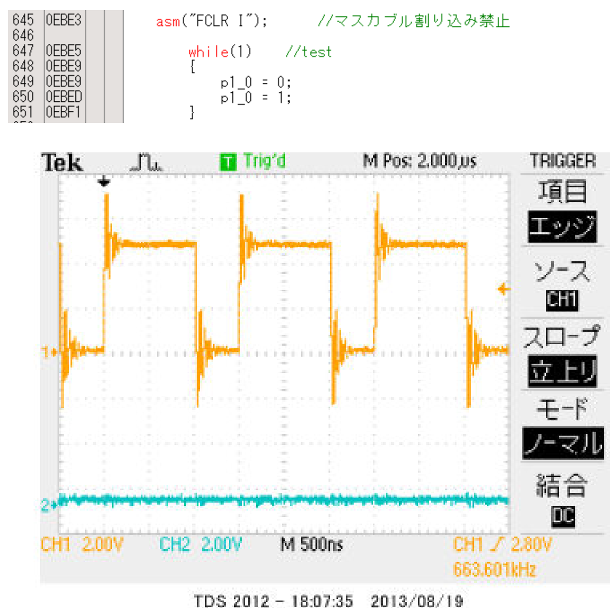
Ｈ８／３００Ｈコアを代表してＨ８／３６１０９を使用しました。基板名ＢＣＨ８３６１４０９。ＨＥＷで同じ意味のコードを書き込みテストします。Ｈ８／３００ＨコアはＨ８／３０４８やＨ８／３０５２と同じです。



ポートＥを繰り返し、０、１しています。波形を観測すると８２８．０６７ＫＨｚとなりました。

６．３８７３２ＭＨｚ÷８２８．０６７ＫＨｚ≒７．７倍高速という驚きの結果になりました。（クロック２０ＭＨｚ）クロックを同じにしても、４．８倍違います。

次にＲ８Ｃを評価します。Ｒ８Ｃ/Ｍ１２Ａ（クロック２０ＭＨｚ）を使用して比較してみます。



６６３．６０１ＫＨｚとなりました。

ｆ－２　乗除演算速度の比較

演算速度はどの程度違うでしょうか？　３２ｂｉｔの乗算、除算を行ってみました。
演算前にポートを立てて、演算後にポートを下ろすことにより、演算実行時間をオシロで観測しています。

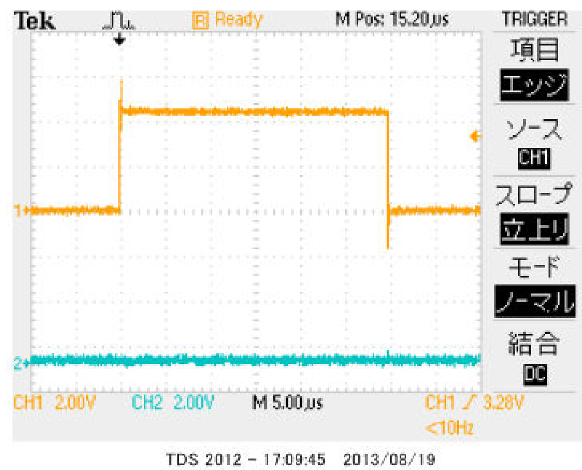
Ｈ８－３６１０９　約３０μｓｅｃでした。

```
        while(1)
        {

            ldata1 = 1234;
            ldata2 = 5678;

            PDRE = 0xff;
            ldata3 = ldata1 * ldata2;
            ldata4 = ldata2 / ldata1;
            PDRE = 0;
/*
            PDR1 = 0;
            PDR2 = 0;
            PDR3 = 0;
            PDR5 = 0;
            PDR7 = 0;
//          PDR8 = 0;
            PDRC = 0;
            PDRD = 0;
```

Address	+0	+4	+8	+C
FFE40C	006AE9BC	00000004	00000000	00000000



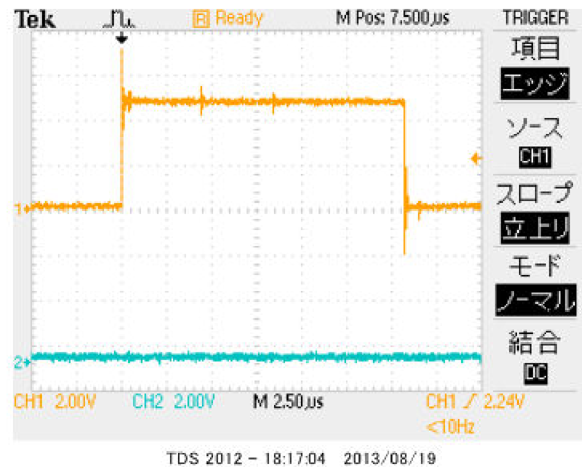
TDS 2012 － 17:09:45　2013/08/19

Ｒ８Ｃ／Ｍ１２Ａの場合　約１５．５μｓｅｃでした。

```
    asm("FCLR I");      //マスカブル割り込み禁止

        while(1)    //test
        {
            ldata1 = 1234;
            ldata2 = 5678;

            p1_0 = 1;
            ldata3 = ldata1 * ldata2;
            ldata4 = ldata2 / ldata1;
            p1_0 = 0;
        }
```



TDS 2012 － 18:17:04　2013/08/19

ＲＬ７８の場合　約３．８μｓｅｃでした。

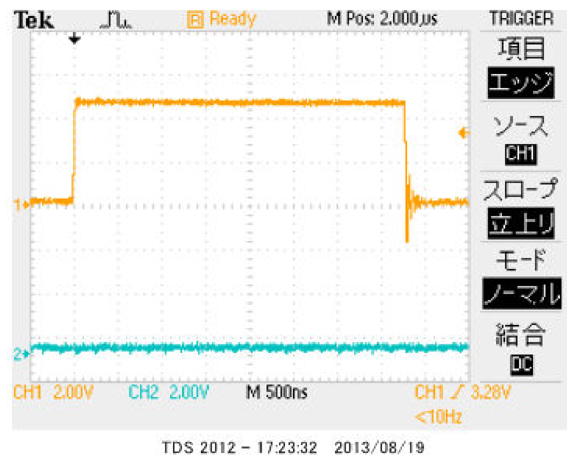
ソースファイル

```
                unsigned long ldata1,ldata2,ldata3,ldata4;

                void main(void)
                {
001f1               R_MAIN_UserInit();
                    /* Start user code. Do not edit comment generated here */
                    while (1U)
                    {
001f5                   ldata1 = 1234;
001ff                   ldata2 = 5678;
00209                   P2 = 0xff;
0020c                   ldata3 = ldata1 * ldata2;
0022b                   ldata4 = ldata2 / ldata1;
0024a                   P2 = 0x0;
```

ソース＋逆アセンブラ

```
73:              ldata1 = 1234;
  001f5  30d204    MOVW     AX,#4D2H
  001f8  bfbaef    MOVW     !_ldata1,AX
  001fb  f6        CLRW     AX
  001fc  bfbcef    MOVW     !0EFBCH,AX
74:              ldata2 = 5678;
  001ff  302e16    MOVW     AX,#162EH
  00202  bfbeef    MOVW     !_ldata2,AX
  00205  f6        CLRW     AX
  00206  bfc0ef    MOVW     !0EFC0H,AX
75:              P2 = 0xff;
  00209  cd02ff    MOV      P2,#0FFH
76:              ldata3 = ldata1 * ldata2;
  0020c  afbaef    MOVW     AX,!_ldata1
  0020f  bdd8      MOVW     _@STBEG,AX
  00211  afbcef    MOVW     AX,!0EFBCH
  00214  bdda      MOVW     _@RTARG2,AX
  00216  afbeef    MOVW     AX,!_ldata2
  00219  bddc      MOVW     _@RTARG4,AX
  0021b  afc0ef    MOVW     AX,!0EFC0H
  0021e  fd8501    CALL     !@@lsmul
  00221  adda      MOVW     AX,_@RTARG2
  00223  bfc4ef    MOVW     !0EFC4H,AX
  00226  add8      MOVW     AX,_@STBEG
  00228  bfc2ef    MOVW     !_ldata3,AX
77:              ldata4 = ldata2 / ldata1;
  0022b  afbeef    MOVW     AX,!_ldata2
  0022e  bdd8      MOVW     _@STBEG,AX
  00230  afc0ef    MOVW     AX,!0EFC0H
  00233  bdda      MOVW     _@RTARG2,AX
  00235  afbaef    MOVW     AX,!_ldata1
  00238  bddc      MOVW     _@RTARG4,AX
  0023a  afbcef    MOVW     AX,!0EFBCH
  0023d  fdb001    CALL     !@@ludiv
  00240  adda      MOVW     AX,_@RTARG2
  00242  bfc8ef    MOVW     !0EFC8H,AX
  00245  add8      MOVW     AX,_@STBEG
  00247  bfc6ef    MOVW     !_ldata4,AX
78:              P2 = 0x0;
  0024a  f402      CLRB     P2
```



以上の結果をまとめると

ＣＰＵコア	クロック	ポートアクセス	乗除演算
ＲＬ７８	３２ＭＨｚ	６．３８ＭＨｚ	３．８μｓｅｃ
Ｈ８－３００Ｈ	２０ＭＨｚ	０．８２ＭＨｚ	３０μｓｅｃ
Ｒ８Ｃ	２０ＭＨｚ	０．６６ＭＨｚ	１５．５μｓｅｃ
結論		ＲＬ７８がＨ８－３００Ｈの７．７倍、Ｒ８Ｃの９．６倍高速。	ＲＬ７８がＨ８－３００Ｈの７．８倍、Ｒ８Ｃの４倍高速。

※測定結果はいずれも弊社製品比較です。

一般に設計が新しいCPUの方が、製造プロセスが微細化されている分、同じ機能であれば安価に製造できます。RL78は従来より優れたアーキテクチャのコアに、積和演算対応、10進補正回路等、高度な機能も内蔵し、かつ、今までより低消費電力、安価を目指して開発されたようです。

結論として、従来、Ｈ８／３０４８等をご使用の方々にも安心して使っていただける性能をもったＣＰＵだと思います。

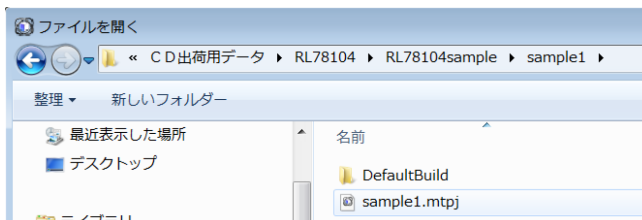
１－２　　動作、デバック

ａ：ＣｕｂｅＳｕｉｔｅ＋起動、コンパイル、書き込み、動作

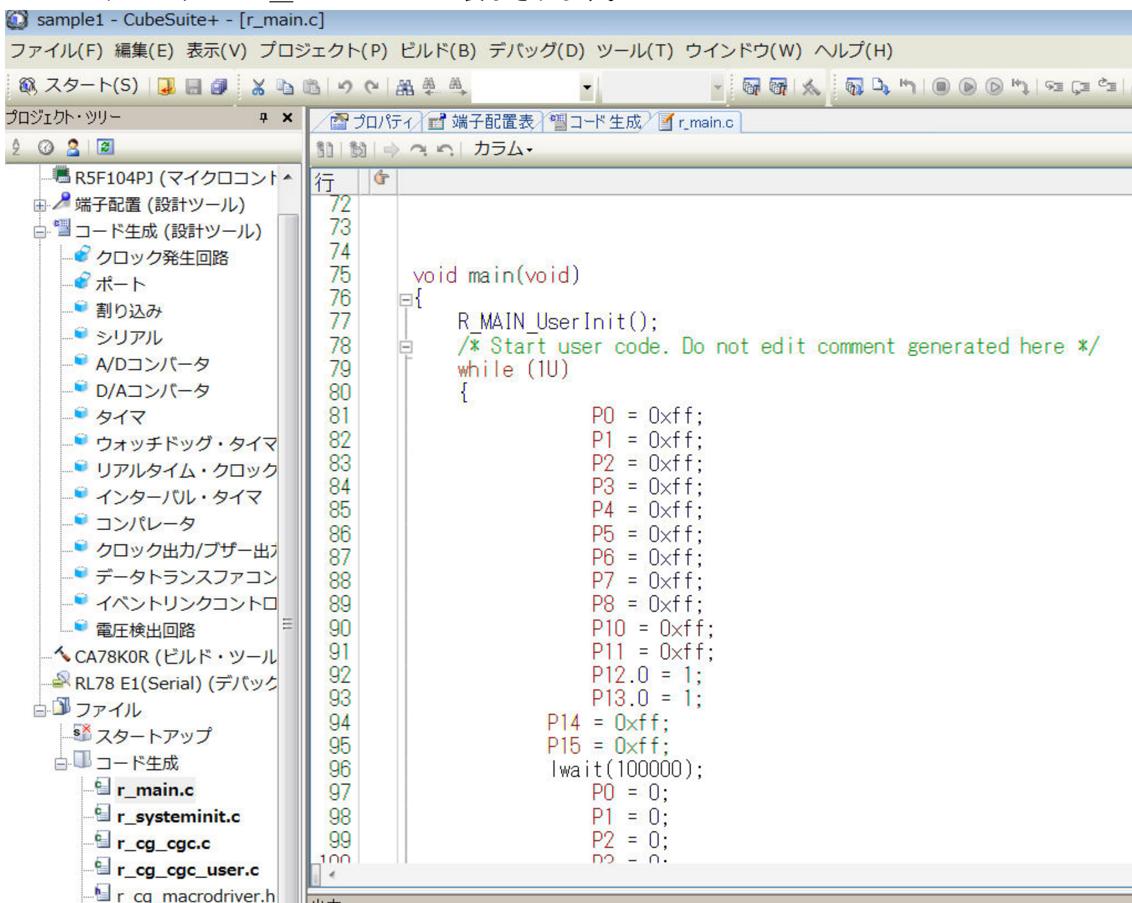


ＣＤに添付しているサンプルプログラムを使って、コンパイル、書き込み、動作の方法を示します。

ＣｕｂｅＳｕｉｔｅ＋（以降ＣＳ+）を起動します。ここでは例としてＲＬ７８１０４￥ｓａｍｐｌｅ１を動作させます。基板上のＬＥＤ　Ｄ１が点滅するプログラムです。
初めてのときは　ファイル　→　ファイルを開く　→　ｓａｍｐｌｅ１．ｍｔｐｊをダブルクリックします。



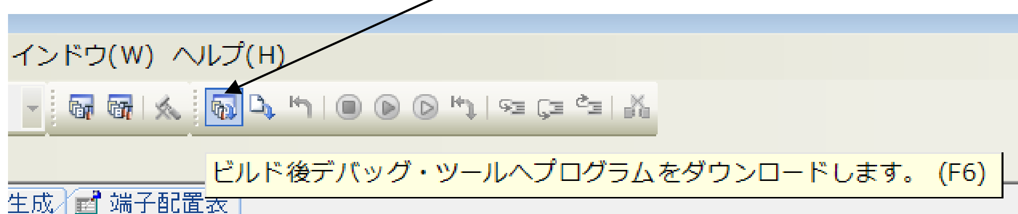
プロジェクトツリーとｒ＿ｍａｉｎ．ｃが表示されます。



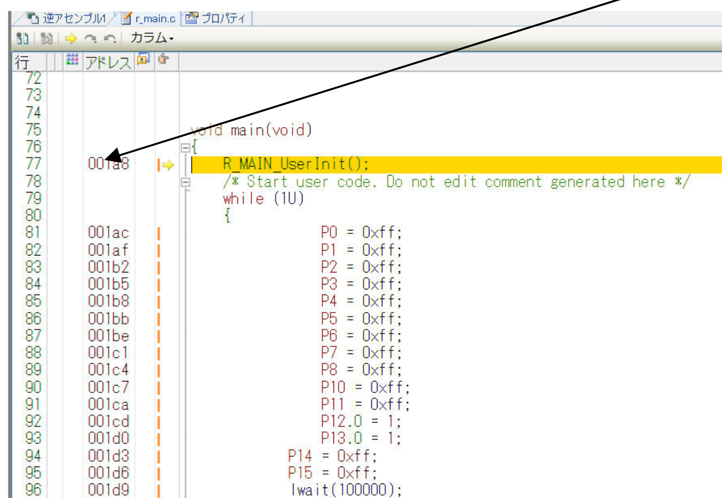
とりあえず、実行してみます。Ｅ１のケーブルを基板のＣＮ１に挿入します。電源はＥ１から供給しますので、不要です。（写真ご参考）



「ビルド後、デバック・ツールへプログラムを転送」をクリック。

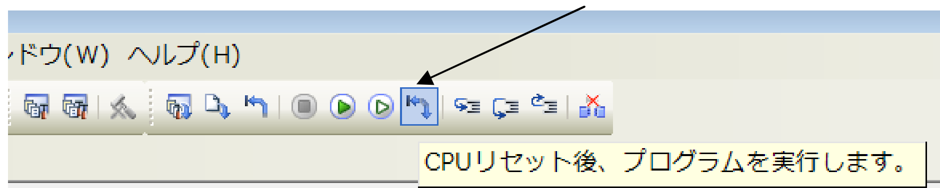


上手く転送できると、今まで表示されていなかったプログラムの絶対アドレスが表示されます。Ｅ１から電源がＣＰＵ基板に供給されます。Ｅ１のＶＣＣの　　　　ＬＥＤの色が緑からオレンジに変わります。



ここまでいかなかった場合、Ｅ１のインストゥールをご検証願います。

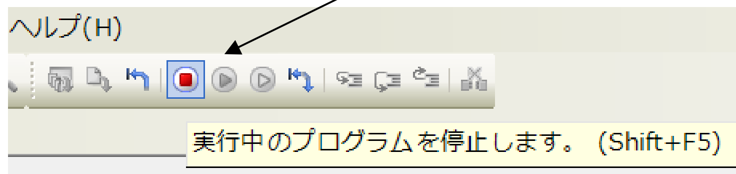
次に、プログラムを動作させせます。「ＣＰＵリセット後、プログラムを実行」をクリック。



Ｅ１のＲＵＮ（青ＬＥＤ）が点灯し、基板のＤ１が点滅したら動作しています。ＣＳ＋の右下部にも表示されます。



ここまで確認できましたら、一度止めます。

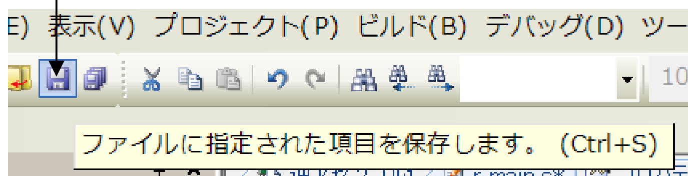


ｍａｉｎ関数のｌｗａｉｔの数値２箇所をキーボードを押して１桁０を増やしてみます。

```
        P0 = 0xff;
        P1 = 0xff;
        P2 = 0xff;
        P3 = 0xff;
        P4 = 0xff;
        P5 = 0xff;
        P6 = 0xff;
        P7 = 0xff;
        P8 = 0xff;
        P10 = 0xff;
        P11 = 0xff;
        P12.0 = 1;
        P13.0 = 1;
    P14 = 0xff;
    P15 = 0xff;
    lwait(1000000);
        P0 = 0;
        P1 = 0;
        P2 = 0;
        P3 = 0;
        P4 = 0;
        P5 = 0;
        P6 = 0;
        P7 = 0;
        P8 = 0;
        P10 = 0;
        P11 = 0;
        P12.0 = 0;
        P13.0 = 0;
    P14 = 0;
    P15 = 0;

    lwait(1000000);
```

セーブして



さきほどの、

「ビルド後、デバック・ツールへプログラムを転送」をクリック。

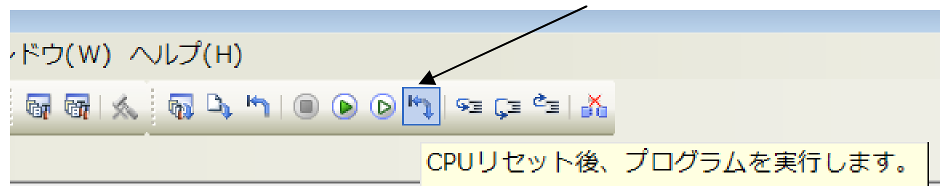
「ＣＰＵリセット後、プログラムを実行」をクリック。

ＬＥＤの点滅が先ほどより、遅くなったのが目視できましたでしょうか？

次に、ブレークポイントの設定を行ってみます。一度、プログラムを停止させます。
ブレークポイントを２点設定しました、手のマーク。設定はカーソルを行にもっていき、右クリック。設定後、右クリックで解除。　　　　　　　　　黄色が現在のプログラム　カウンタ位置。

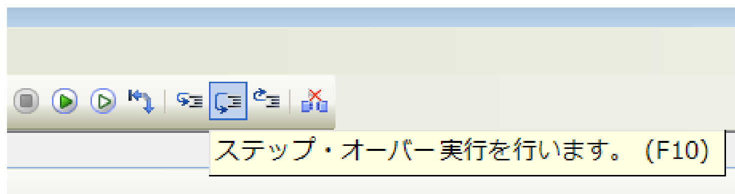
```
              /* Start user code. Do not edi
              while (1U)
              {
001ac                 P0 = 0xff;
001af                 P1 = 0xff;
001b2                 P2 = 0xff;
001b5                 P3 = 0xff;
001b8                 P4 = 0xff;
001bb                 P5 = 0xff;
001be                 P6 = 0xff;
001c1                 P7 = 0xff;
001c4                 P8 = 0xff;
001c7                 P10 = 0xff;
001ca                 P11 = 0xff;
001cd                 P12.0 = 1;
001d0                 P13.0 = 1;
001d3             P14 = 0xff;
001d6             P15 = 0xff;
001d9             lwait(100000);
001e1                 P0 = 0;
001e3                 P1 = 0;
001e5                 P2 = 0;
001e7                 P3 = 0;
001e9                 P4 = 0;
001eb                 P5 = 0;
001ed                 P6 = 0;
001ef                 P7 = 0;
001f1                 P8 = 0;
001f3                 P10 = 0;
001f5                 P11 = 0;
001f7                 P12.0 = 0;
001fa                 P13.0 = 0;
001fd             P14 = 0;
001ff             P15 = 0;
```

プログラムを動作させます。「ＣＰＵリセット後、プログラムを実行」をクリック。



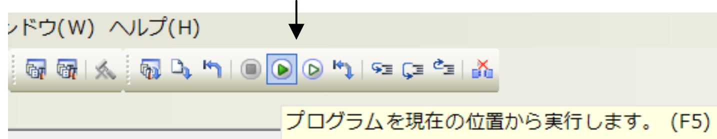

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
                P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P4 = 0xff;
                P5 = 0xff;
                P6 = 0xff;
                P7 = 0xff;
                P8 = 0xff;
                P10 = 0xff;
                P11 = 0xff;
                P12.0 = 1;
                P13.0 = 1;
            P14 = 0xff;
            P15 = 0xff;
            lwait(100000);
```

プログラムの実行はブレークポイントで停止します。



ステップオーバー実行をクリックし、１行進めます。
ＬＥＤ　Ｄ１　は　Ｐ１４＝０ｘｆｆ；命令により点灯します。

「プログラムを現在の位置から実行」します。




```
        P13.0 = 1;
    P14 = 0xff;
    P15 = 0xff;
    lwait(100000);
        P0 = 0;
        P1 = 0;
        P2 = 0;
        P3 = 0;
        P4 = 0;
        P5 = 0;
        P6 = 0;
        P7 = 0;
        P8 = 0;
        P10 = 0;
        P11 = 0;
        P12.0 = 0;
        P13.0 = 0;
    P14 = 0;
    P15 = 0;

    lwait(100000);
    ;
}
/* End user code. Do not edit comment generated here */
```

次のブレークポイント設定行　Ｐ１４＝０でプログラムは停止します。この行はまだ実行されていません。
ステップオーバー実行をクリックし、１行進めます。
ＬＥＤの消灯が確認できると思います。

```
    P14 = 0xff;
    P15 = 0xff;
    lwait(100000);
        P0 = 0;
        P1 = 0;
        P2 = 0;
        P3 = 0;
        P4 = 0;
        P5 = 0;
        P6 = 0;
        P7 = 0;
        P8 = 0;
        P10 = 0;
        P11 = 0;
        P12.0 = 0;
        P13.0 = 0;
    P14 = 0;
    P15 = 0;

    lwait(100000);
    ;
}
/* End user code. Do not edit comment generated here */
```

以上が、プログラムのコンパイル、Ｅ１へのダウンロード、実行、修正、ブレークポイント設定、動作の概要です。

ｂ：新しいプログラムを作る

以下省略

ｂ－２：自動生成されたプログラム

生成されたプログラム抜粋ですが

関数名	動作
ｒ＿ｍａｉｎ．ｃ	ｍａｉｎ関数（）があり、ユーザーはここにアプリケーションプログラムを書きます。
ｒ＿ｓｙｓｔｅｍｉｎｉｔ．ｃ	初期化コードが自動生成されていて、電源ＯＮ時に自動実行されます。
ｒ＿ｃｇ＿ｃｇｃ．ｃ	自動生成されたクロックプログラム。ｒ＿ｓｙｓｔｅｍｉｍｉｔ．ｃから呼ばれるｘｘＣｒｅａｔｅ（）関数があります。
ｒ＿ｃｇ＿ｃｇｃ＿ｕｓｅｒ．ｃ	自動生成されたクロックプログラム ユーザーが内容を書き換えて使用できます。
..	..
ｒ＿ｃｇ＿ｓｅｒｉａｌ．ｃ	ＵＡＲＴ３　Ｃｒｅａｔｅ（）関数があります。 電源ＯＮ時、ｒ＿ｓｙｓｔｅｍｉｍｉｔ．ｃから自動的に呼ばれます。 （初期化をユーザーは意識する必要がありません） 動作を開始させるためのＳｔａｒｔ（）関数があります。 Ｓｔａｒｔ（）関数は動作開始時、ｍａｉｎルーチンにユーザーが書き込む必要があります。
ｒ＿ｃｇ＿ｓｅｒｉａｌ＿ｕｓｅｒ．ｃ	ＵＡＲＴ３　の送受信関数群が作成されています。
..	

ペリフェラル毎に、ファイル名にｕｓｅｒが付くのと付かないＣファイルがセットで自動生成されていて、付かないほうは初期設定、スタート関数など、ユーザーは手を加えない、付くほうはユーザーが書き加えてプログラムを完成させるようになっています。

各々の関数説明は後の個別のサンプルプログラムで記述します。ユーザーはプログラムを主に、ｒ＿ｍａｉｎ．ｃの中のｍａｉｎ（）関数の中にｍａｉｎプログラムを記述します。ｒ＿ｓｙｓｔｅｍｉｎｉｔ．ｃは各使用するペリフェラル（ＩＯやＳＩＯ等）の初期化ルーチンです。電源ＯＮ時に自動実行されますので、ユーザーは意識する必要がありません。

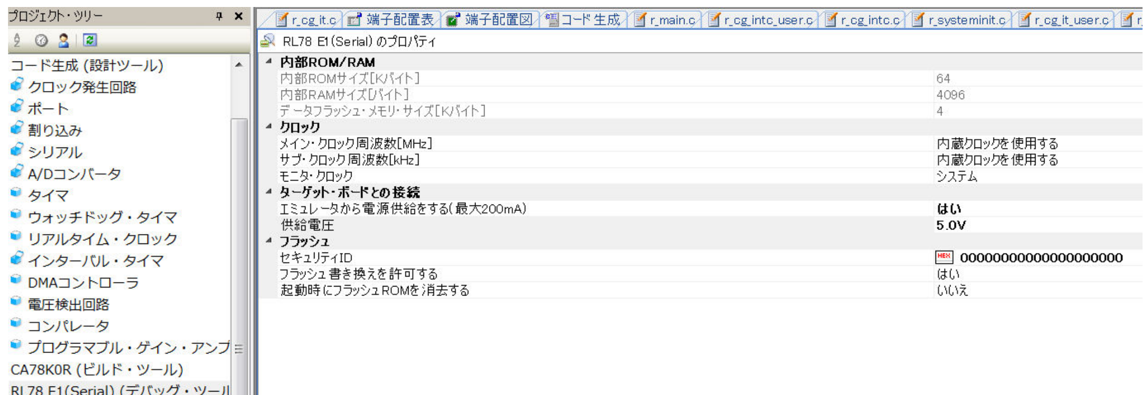
Ｒ＿Ｓｙｓｔｅｍｉｎｉｔ（ｖｏｉｄ）関数の中身

```
void R_Systeminit(void)
{
    PIOR0 = 0x00U;
    PIOR1 = 0x00U;
    R_CGC_Get_ResetSource();
    R_PORT_Create();
    R_CGC_Create();
    R_SAU1_Create();
    R_ADC_Create();
    CRC0CTL = 0x00U;
    IAWCTL = 0x00U;
    PMS = 0x00U;
}
```

Ｒ＿ＰＯＲＴ＿Ｃｒｅａｔｅ（）；　　　//ポート初期化
Ｒ＿ＣＧＣ＿Ｃｒｅａｔｅ（）；　　　　//クロック初期化
Ｒ＿ＳＡＵ１＿Ｃｒｅａｔｅ（）；　　　//ＵＡＲＴ初期化
Ｒ＿ＡＤＣ＿Ｃｒｅａｔｅ（）；　　　　//Ａ／Ｄコンバータ初期化
；
などが自動生成され、電源ＯＮ時に自動的に実行されます。

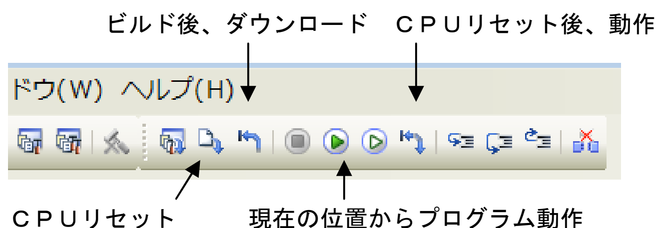
ｂ－３：Ｅ１から電源供給

デバッカにはＥ１を使用します。Ｅ１から電源を供給する設定は



ＲＬ７８Ｅ１（Ｓｅｒｉａｌ）（デバックツール）を右クリック→プロパティで上記画面になりますので、エミュレータから電源を供給するを「はい」、電圧を「５Ｖ」にして下さい。３．３Ｖでも問題なく動作します。外部電源を使用する場合、ダウンロード前に電源をＯＮさせる必要があります。

ビルド後、ダウンロードを行いＥ１とうまく通信が出来るとデバックのためのボタンがアクティブになります。



ＣＰＵリセット後、動作をクリックするとプログラムが初めから動作します。

ｂ－４：コード生成後の初期値の変更

「コード生成」後、プログラムをある程度書いた後の仕様の変更に、再び「コード生成」を行うと、既にプログラムを書いた部分が初期化されてしまう場合があります。そこで、「コード生成」を使わずに、Ｃｒｅａｔｅ（）関数の中を変更する例を示します。例はＳＩＯボーレート９６００ｂｐｓを３８４００ｂｐｓに変更した例です。

r_cg_serial.c - メモ帳

ファイル(F)　編集(E)　書式(O)　表示(V)　ヘルプ(H)

```
/*****************************************************************************************
* Function Name: R_SAU0_Create
* Description  : This function initializes the SAU0 module.
* Arguments    : None
* Return Value : None
******************************************************************************************
void R_SAU0_Create(void)
{
    SAU0EN = 1U;    /* supply SAU0 clock */
    NOP();
    NOP();
    NOP();
    NOP();
//    SPS0 = _0004_SAU_CK00_FCLK_4 | _0040_SAU_CK01_FCLK_4;            //9800bps
    SPS0 = _0002_SAU_CK00_FCLK_2 | _0020_SAU_CK01_FCLK_2;            //38400bps
    R_UART1_Create();
}

/*****************************************************************************************
* Function Name: R_UART1_Create
* Description  : This function initializes the UART1 module.
* Arguments    : None
* Return Value : None
******************************************************************************************
void R_UART1_Create(void)
{
```

ｒ＿ｃｇ＿ｓｅｒｉａｌ．ｈの中に定義があります。

```
/*
    Serial Clock Select Register m (SPSm)
*/
/* Operating mode and clear mode selection (PRS003 - PRS000) */
#define _0000_SAU_CK00_FCLK_0            (0x0000U)    /* ck00 - fCLK */
#define _0001_SAU_CK00_FCLK_1            (0x0001U)    /* ck00 - fCLK/2^1 */
#define _0002_SAU_CK00_FCLK_2            (0x0002U)    /* ck00 - fCLK/2^2 */
#define _0003_SAU_CK00_FCLK_3            (0x0003U)    /* ck00 - fCLK/2^3 */
#define _0004_SAU_CK00_FCLK_4            (0x0004U)    /* ck00 - fCLK/2^4 */
```

９６００ｂｐｓと３８４００では速度が４倍違いますから、元の＿０００４を＿０００２に変えます。

b－５：変数を見る

```
}

R_ADC_Start();                          //AD変換開始
R_ADC_Get_Result(ad_data)

P3.0 = 1;
P3.0 = 0;
```

ウォッチ1 に登録(R)
解析グラフ に登録(E)
アクション・イベントの登録(A)...
切り取り(T)　Ctrl+X

見たい変数をコピーして右クリック→ウオッチ１に登録

ウォッチ1

表記(N)

ウォッチ式	値	型情報(バイト数)	アドレス	メモ
ad_data	0 (0x0000)	unsigned short(2)	0xfefd0	

b－６：変数変化を実行中に確認する

ウォッチ1

表記(N)

ウォッチ式	値	型情報(バイト数)	アドレス	メモ
ad_data	0 (0x0000)	unsigned short(2)	0xfefd0	
ADCS	0x0	SFR[R/W 1](1ビッ…	0xfff30.7	
ADM0	0x00	SFR[R/W 1.8](1)	0xfff30	
ADM1	0x20	SFR[R/W 1.8](1)	0xfff32	
ADM2	0x00	SFR[R/W 1.8](1)	0xf0010	
loop	??		?	
ad_buffer	""	unsigned char…	0xfefd4	
enc_p	0 (0x0000)	unsigned short(2)	0xfefe8	
enc_m	0 (0x0000)	unsigned short(2)	0xfefea	
PM20	0x80	SFR[R/W 1.8](1)	0xf0510	

ウオッチウインドウはデバックに非常に便利な窓ですが、そのままでは動作中は更新されません。そこで、

プロジェクト・ツリー

- ba2_3chPWM_RL78 (プロジェク
  - R5F107DE (マイクロコントロー
  - 端子配置 (設計ツール)
  - コード生成 (設計ツール)
  - CA78K0R (ビルド・ツール)
  - RL78 E1(Serial) (デバッグ・ツ
  - プログラム解析 (解析ツール)
  - ファイル
    - ビルド・ツール生成ファイル
    - スタートアップ
    - コード生成
      - r_main.c

逆アセンブル1 | r_main.c | r_cg_intc_user.c | r_cg_intc.c | r_cg_it.c | r_systeminit.c | r_cg

RL78 E1(Serial) のプロパティ

メモリ	
メモリ・マッピング	[9]
メモリ書き込み時にベリファイを行う	はい
実行中のメモリ・アクセス	
実行を一瞬停止してアクセスする	**はい**
実行中に表示更新を行う	はい
表示更新間隔[ms]	500
ブレーク	
入力信号のマスク	
TARGET RESET 信号をマスクする	いいえ
INTERNAL RESET 信号をマスクする	いいえ

ＲＬ７８　Ｅ１（Ｓｅｒｉａｌ）（デバック・ツール）→プロパティ→デバックツール設定で実行中のメモリアクセスを「はい」にすると、実行中でも変数の変化が確認できます。

下記例はｓｐｒｉｎｔｆでａｄ＿ｂｕｆｆにｅｅｐ＿ｄａｔａの値が格納されるのをリアルタイムで表示しています。

intc_user.c | r_cg_intc.c | r_cg_it.c | r_systeminit.c | r_cg_it_user.c | r_cg_serial.c | プロパティ

```
prom test

            eep_init();

            eep_wr16(0,250);                          //TEMP          25.0℃
            eep_wr16(2,500);                          //THICK         500nm
            eep_wr16(4,100);                          //CAL           1.00

            eep_data = eep_rd16(0);
            sprintf(ad_buffer, "eep = %4d¥n¥r",eep_data);   //ad_buffに10進ASCII
            R_UART1_Send(ad_buffer,sizeof(ad_buffer));      //uart出力
            tx_end_wait();


    while(1)
            ;


/* Start user code. Do not edit comment generated here */
while (1U)
{
```

ウォッチ1

表記(N)

ウォッチ式	値	型情報(バイト数)	アドレス
ad_data	0 (0x0000)	unsigned short(2)	0xfefd0
ADCS	0x0	SFR[R/W 1](1ビッ…	0xfff30.7
ADM0	0x01	SFR[R/W 1.8](1)	0xfff30
ADM1	0x20	SFR[R/W 1.8](1)	0xfff32
ADM2	0x00	SFR[R/W 1.8](1)	0xf0010
loop	?	?	?
ad_buffer	""	unsigned char…	0xfefd4
[0]	'e' (0x65)	unsigned char(1)	0xfefd4
[1]	'e' (0x65)	unsigned char(1)	0xfefd5
[2]	'p' (0x70)	unsigned char(1)	0xfefd6
[3]	' ' (0x20)	unsigned char(1)	0xfefd7
[4]	'=' (0x3d)	unsigned char(1)	0xfefd8
[5]	' ' (0x20)	unsigned char(1)	0xfefd9
[6]	' ' (0x20)	unsigned char(1)	0xfefda
[7]	'2' (0x32)	unsigned char(1)	0xfefdb
[8]	'5' (0x35)	unsigned char(1)	0xfefdc
[9]	'0' (0x30)	unsigned char(1)	0xfefdd
[10]	'' (0x0a)	unsigned char(1)	0xfefde
[11]	'' (0x0d)	unsigned char(1)	0xfefdf
[12]	'' (0x00)	unsigned char(1)	0xfefe0
[13]	'' (0x00)	unsigned char(1)	0xfefe1
[14]	'' (0x00)	unsigned char(1)	0xfefe2
[15]	'' (0x00)	unsigned char(1)	0xfefe3
[16]	'' (0x00)	unsigned char(1)	0xfefe4
[17]	'' (0x00)	unsigned char(1)	0xfefe5

なお、使用端子や動作プログラムが同じような構成のものの場合、ホルダをコピーし、ホルダ名、ｍｔｐｊファイルの名前を変更すれば、それまでの設定はそのまま使えます。変更もその上から行うことが出来ます。

２．サンプルプログラム

２－１　ｓａｍｐｌｅ１　出力ポートのＯＮ，ＯＦＦ

```
/******************************************************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
******************************************************************************************/

①void lwait(unsigned long time)
{
        while(time != 0)
        {
                time--;
        }

}



②void main(void)
{
   ③ R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
   ④ while (1U)
    {
                P0 = 0xff;
                P1 = 0xff;
                P2 = 0xff;
                P3 = 0xff;
                P4 = 0xff;
                P5 = 0xff;
                P6 = 0xff;
                P7 = 0xff;
                P8 = 0xff;
                P10 = 0xff;
                P11 = 0xff;
                P12.0 = 1;
                P13.0 = 1;
    ⑤       P14 = 0xff;
            P15 = 0xff;
    ⑥       lwait(100000);
                P0 = 0;
```

```
                P1 = 0;
                P2 = 0;
                P3 = 0;
                P4 = 0;
                P5 = 0;
                P6 = 0;
                P7 = 0;
                P8 = 0;
                P10 = 0;
                P11 = 0;
                P12.0 = 0;
                P13.0 = 0;
⑦           P14 = 0;
            P15 = 0;

⑧           lwait(100000);
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

【　解説　】

①void lwait(unsigned long time)

下のｍａｉｎ関数から呼ばれるウエイトルーチンです。

②void main(void)

{

メインルーチンです。

③ R_MAIN_UserInit();

コード生成によって自動的に作られた初期設定関数をコールしています。この初期設定はメインルーチンの下にあります。割込み許可ＥＩを実行しているだけです。

```
    /* Start user code. Do not edit comment generated here */
④ while (1U)
    {
```

以下を無限ループします。

⑤P14 = 0xff;

Ｐ１４に０ｘｆｆを設定しています。Ｐ０の出力設定は、コード生成により、ｒ＿ｓｙｓｔｅｍｉｎｉｔ．ｃの中のＲ￥ｓｙｓｔｅｍｉｎｉｔ（）関数の中にあり、リセット解除後、自動実行されます。

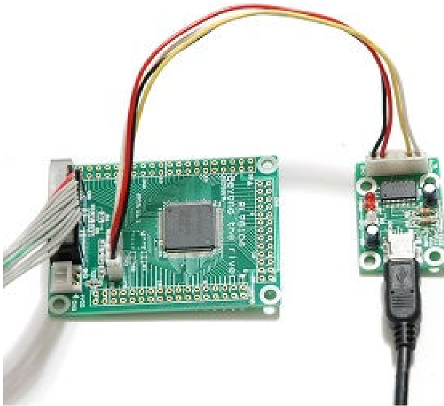
出力に設定されたＰ１４のポートは全て１になります。よってＰ１４５も１（電源電圧、５Ｖの場合、約５Ｖ、３．３Ｖの場合、約３．３Ｖ）が出力され、接続されているＬＥＤ　Ｄ１に電流が流れ点灯します。

⑥　　　　　lwait(100000);
設定された数が０になるまでループするウエイト関数です。

⑦　　　　　P14 = 0;
Ｐ１４に０を設定しています。Ｐ１４５に接続されているＬＥＤ　Ｄ１は消灯します。

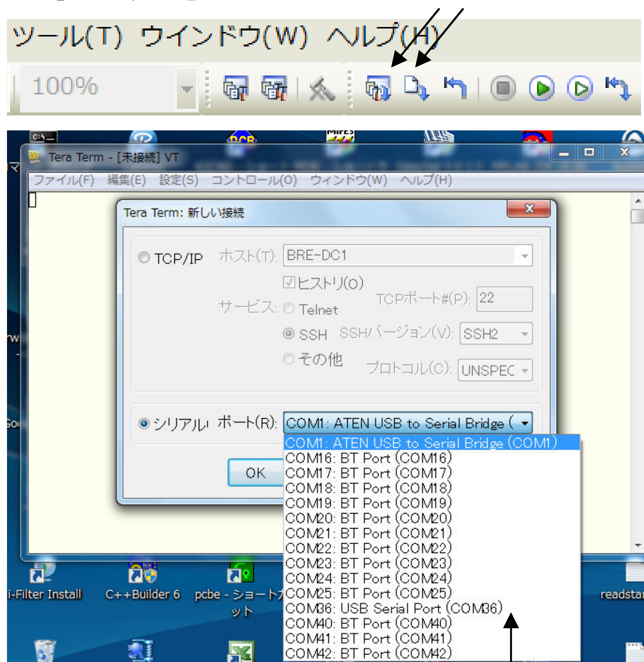
⑧　　　　　lwait(100000);
点灯も消灯と同じ時間、保持されます。

２－２　ｓａｍｐｌｅ２　ＳＩＯ（ＵＳＢ)、ＥＥＰＲＯＭ読み書き



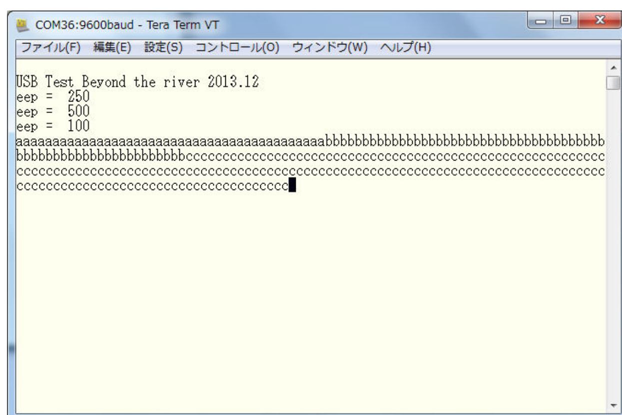
【　概要　】

ＵＳＢ出力をパソコンと接続し、データのやり取りを行います。添付のＶケーブルをＣＰＵボードのＣＮ３に接続します。ＵＳＢミニケーブルをパソコンと接続します。お手数ですが、無料のターミナルプログラム、テラタームやハイパーターミナルなどのターミナルプログラムを使用しますので、無い方は、ネットで検索し、インストゥール願います。例ではテラタームで行います。38400bps に設定して下さい。USB ケーブルでパソコンとつなげ、Ｅ１からＣＰＵ基板に電源が入った以降に、30 秒以上経過後テラタームを立ち上げて下さい。(Ｅ１のＶＣＣ　ＬＥＤが点灯以降、例えば「(ビルド後) デバックツールへダウンロード」で基板に電源が入ります)



テラタームの立ち上げでＵＳＢ　Ｓｅｒｉａｌ　Ｐｏｒｔと出てくればＦＴ２３２ＲＬが準備完了です。なお、マイコン基板の電源がＯＦＦになると設定は無効になり、電源ＯＮ時に再び、上記設定が必要になります。

「リセットから実行」で　ｅｅｐ＝１００まで表示されれば正常です。それ以降はパソコンのキーボードを押した文字がＣＰＵ基板に送信され、それを返信（エコーバック）し、表示されるようになっています。

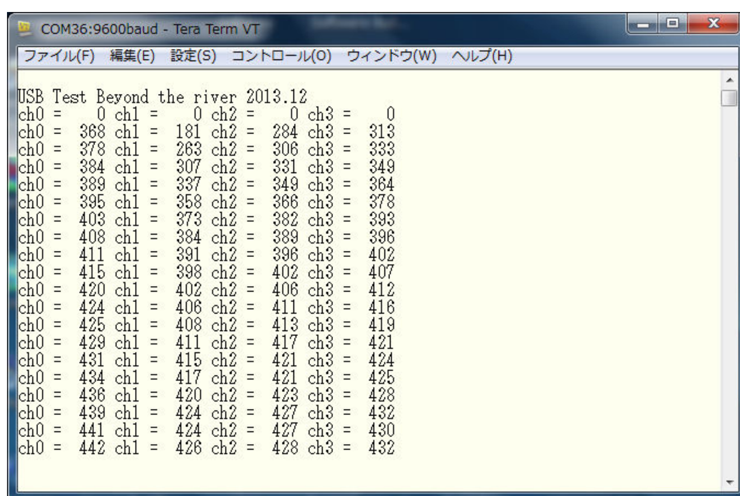


以下省略

２－３　ｓａｍｐｌｅ３　Ａ／Ｄ変換をＵＳＢ出力

【　動作概要　】

ＡＮＩ０，１，２，３を入力とし、Ａ／Ｄ変換した値をＵＳＢからパソコンに送ります。



パソコン側のテラタームではＡＤの数値が繰り返し表示されます。初めの数回は０表示、ＡＮＩｘオープンで０以外、＋５Ｖ接続で１０２３、ＧＮＤ接続で０が表示されます。

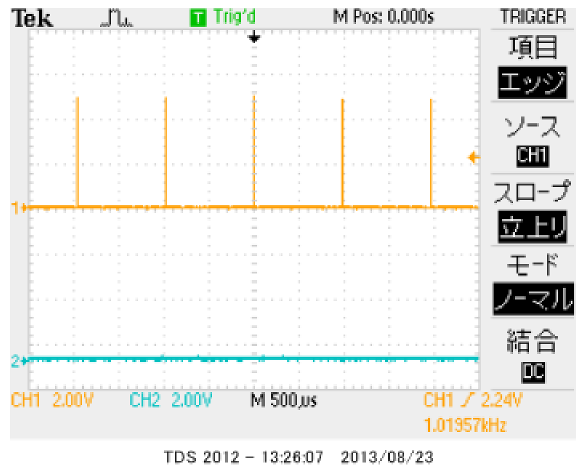
以下省略

。

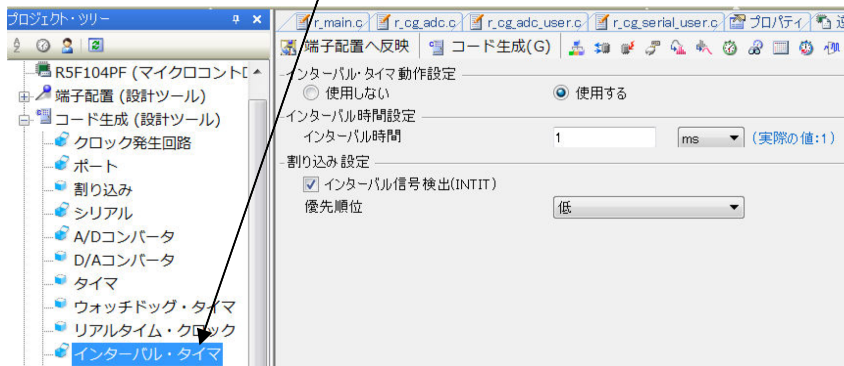
２－４　ｓａｍｐｌｅ４　割り込み

【　動作概要　】

ｓａｍｐｌｅ４を動作させます。オシロスコープがあればＣＮ４　１３番　Ｐ１４５を観測すると、以下のような波形が観測できます。



これはコード生成、インターバルタイマで定周期割り込みを設定したためです。



１ｍｓｅｃ毎に割り込みが入ります。ｒ＿ｃｇ＿ｉｔ＿ｕｓｅｒ．ｃの中に自動的に以下の関数のスケルトンが作成されますから、１ｍｓｅｃに１回実行したいことを書きます。下記例ではＰ０５のＯＮ，ＯＦＦ，タイマーをデクリメントしています。さきほどのオシロで観測された波形はここで作成されています。

```
__interrupt static void r_it_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */

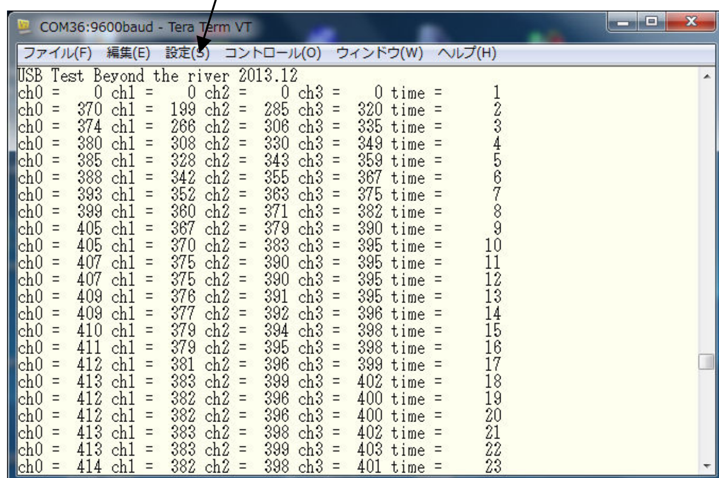
    P14.5 = 1;                          //マーカー

    if(int_time != 0)
    {
        int_time--;
    }

    P14.5 = 0;                          //マーカー
```

```
    /* End user code. Do not edit comment generated here */
}
```

ｍａｉｎではこのｉｎｔｔｉｍｅを使い、ｓａｍｐｌｅ３ではアバウトだった表示時間をちゃんと規定しています。　　　　　１秒毎に表示



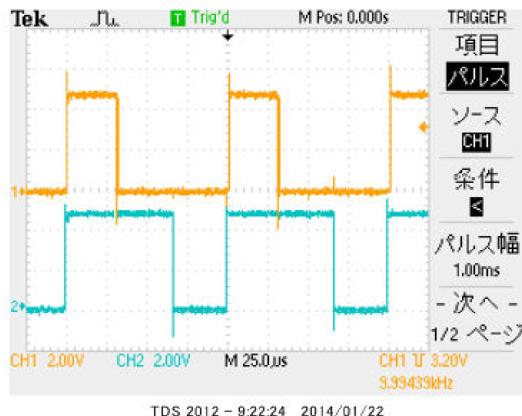
以下省略

２－５　ＰＷＭ出力

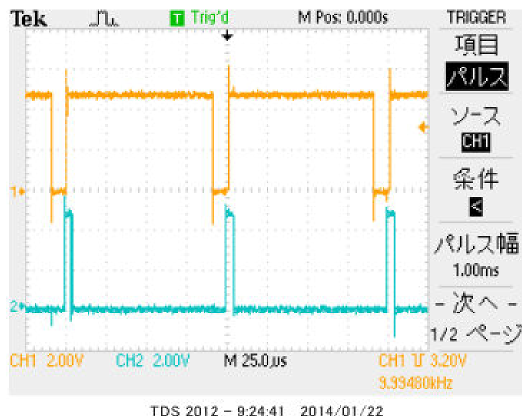
【　動作　】

ＲＬ７８のタイマ・アレイ・ユニットを使用して Pulse-Width Modulation（パルス幅変調）出力を製作します。波形はそれぞれＰ１６（ＴＯ０１　ＣＮ６　１２番）、Ｐ３１（ＴＯ０３　ＣＮ５　２０端子）から出力されます。

波形は下図のように、周期が変わらず、時間経過によってＨ、Ｌの幅の比率が変化します。この出力でＬＥＤやモーターをドライブすると明るさや速度を変えることが出来ます。マイコンと親和性が良い、トランジスタをスイッチとして使用するのでエネルギー効率が良い、などの理由で現代では様々な用途に使われています。



TDS 2012 – 9:22:24　2014/01/22



TDS 2012 – 9:24:41　2014/01/22

プログラムはＰＷＭを出力するために以下の設定になっています。

【　ピン設定　】

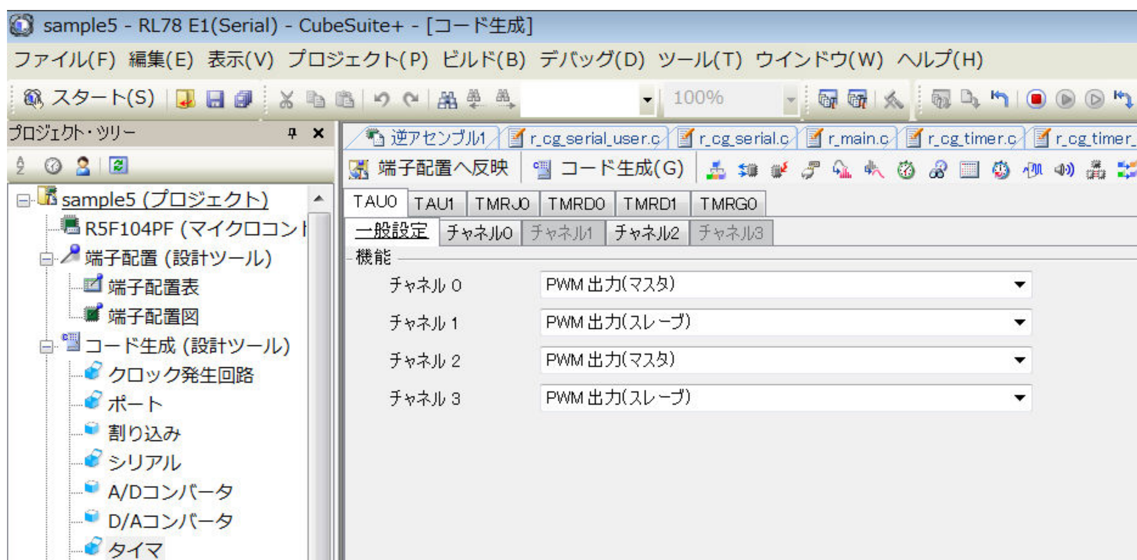
Ｐ１６をＴＯＯ１　１６ビット・タイマ出力に設定。

62	P17/TI02/TO02/TRDIOA0/TRDCLK0/IVCMP0	Free	–	–		
63	P16/TI01/TO01/INTP5/TRDIOC0/IVREF0	TO01	O	–		16ビット・タイマ01 出力
64	P15/_SCK20/SCL20/TRDIOB0	Free	–	–		

Ｐ３１をＴＯＯ３に設定。

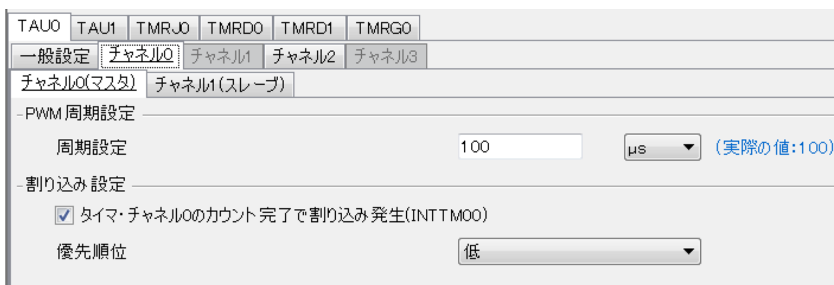
27	P63/SDAA1	Free	–	–		
28	P31/TI03/TO03/INTP4	TO03	O	–		16ビット・タイマ03 出力
29	P64/TI10/TO10	Free	–	–		

【　コード生成設定　】



コード生成→タイマ→ＴＡＵ０を選択、チャンネル０と２をマスタ、チャンネル１と３をスレーブに設定。

チャンネル０マスタの周期を１００μｓｅｃにしてあります。分解能もこれで決まります。



チャンネル１スレーブのデューティ初期値を５０％、出力初期値を０に設定。割り込みはアクティブですが、使用していません。使用しなくても問題ありませんが、製作しておくと使いたいときに使用できます。

TAU0 | TAU1 | TMRJ0 | TMRD0 | TMRD1 | TMRG0

一般設定 | チャネル0 | チャネル1 | チャネル2 | チャネル3

チャネル0(マスタ) | チャネル1(スレーブ)

PWMデューティ設定

デューティ 50 (%) (実際の値:50%)

出力設定

初期出力値 0

出力レベル アクティブ・ハイ

割り込み設定

☑ タイマ・チャネル1のカウント完了で割り込み発生(INTTM01)

優先順位 低

チャンネル２，３も同様の設定です。

【　プログラム　】

以下省略

ウォッチ1

表記(N)

ウォッチ式	値	型情報(バイト数)	アドレス	メモ
TDR03	0x0262	SFR[R/W 16](2)	0xfff66	
TDR01	0x0261	SFR[R/W 16](2)	0xfff1a	
TDR02	0x0c7f	SFR[R/W 16](2)	0xfff64	
TDR00	0x0c7f	SFR[R/W 16](2)	0xfff18	

【　解説　】

従来のツールですと、ＰＷＭを作成する場合、ハードウエアマニュアルで各種レジスタの詳細な理解、初期設定、プログラムが必要でした。ところがＣＳ+の「コード生成」機能を使うと、レジスタに対する詳細な理解、初期設定は必要ありません。本例は、こんなに簡単にＰＷＭを作成できるというＣＳ＋の優位性を示す良い例になると思います。

２－６　三角、対数、平方根関数を使う

【　動作　】

浮動小数点３２ビットｄｏｕｂｌｅを使ってｌｏｇ、ｓｉｎ、√　の演算、キャストを行います。

省略

演算結果ですが、事前予想通りとなりました。

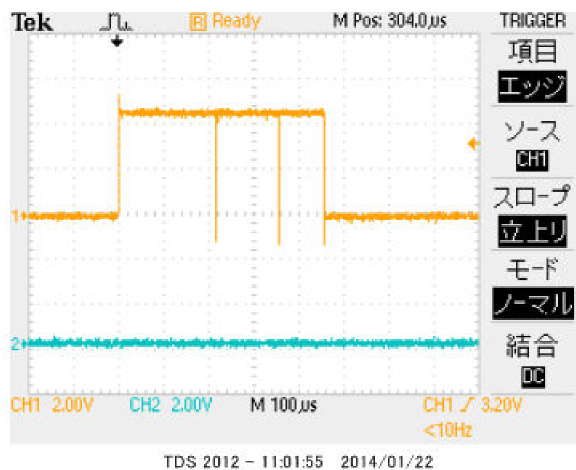
ウォッチ1

表記(N)

ウォッチ式	値	型情報(バイト数)
d1	4.00000 (0x40800000)	double(4)
d2	7.071067691e-1…	double(4)
d3	1.41421 (0x3fb504f3)	double(4)
s1	4 (0x0004)	short(2)
s2	0 (0x0000)	short(2)
s3	1 (0x0001)	short(2)

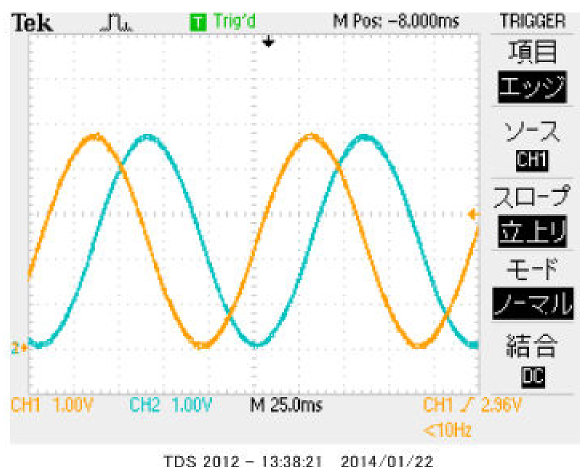
演算速度ですが、

ｌｏｇ１０（１００００）が約２２０μｓｅｃ、ｓｉｎ(４５°）が１３０μｓｅｃ、√２が１００μｓｅｃ程度かかるようでした。



２－７　Ｄ／Ａコンバータ　sin、cos 値を出力してみる

【　動作　】



TDS 2012 - 13:38:21　2014/01/22

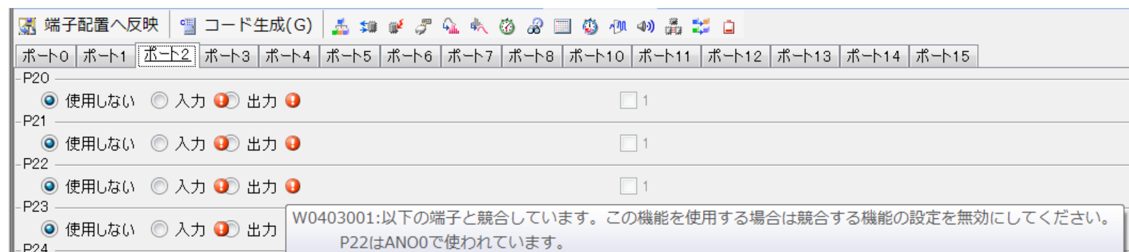
ＲＬ７８１０４は８ビットＤ／Ａ出力を２ｃｈ持っています。そこにｓｉｎ()，ｃｏｓ()の０～３６０°を演算し、Ｄ／Ａ出力し、電圧をみてみます。いわゆる、正弦波発振器と同じ出力が得られます。

【　コード生成設定　】

Ｐ２２はＤ／Ａ０出力　ＡＮＯ０として使用します。Ｐ２３はＤ／Ａ１出力　ＡＮＯ１として使用します。

87	P23/ANI3/ANO1	ANO1	O	-		D/Aコンバータ出力
88	P22/ANI2/ANO0	ANO0	O	-		D/Aコンバータ出力

端子は使用しない、に設定して下さい



【　プログラム　】

以下省略

それぞれはそれぞれの会社の登録商標です。
フォースⓇは弊社の登録商標です。

１．本文章に記載された内容は弊社有限会社ビーリバーエレクトロニクスの調査結果です。
２．本文章に記載された情報の内容、使用結果に対して弊社はいかなる責任も負いません。
３．本文章に記載された情報に誤記等問題がありましたらご一報いただけますと幸いです。
４．本文章は許可なく転載、複製することを堅くお断りいたします。

お問い合わせ先：
〒３５０－１２１３　埼玉県日高市高萩１１４１－１
TEL　０４２（９８５）６９８２
FAX　０４２（９８５）６７２０
HOMEPAGE：http//beriver.co.jp
e-mail：info@beriver.co.jp
有限会社ビーリバーエレクトロニクス　